2 Programming Report

2.1 Decision Tree

2.1.1 Data Preprocessing & Summary Statistics

We first read the data and found there are 345 observations in this dataset. Then, we check the incomplete cells, i.e., NaN and Null, and found 11 observations. Since the missing data include the attribute of sex, it's hard to assign the sex to penguins randomly, hence we drop the observations with empty cells. Below are the summary statistics (Table 2).

Table 2: Summary Statistics

	bill_length_mm	**bill_depth_mm**	**flipper_length_mm**	**body_mass_g**
count	333.000000	333.000000	333.000000	333.000000
mean	43.992793	17.164865	200.966967	4207.057057
std	5.468668	1.969235	14.015765	805.215802
min	32.100000	13.100000	172.000000	2700.000000
25%	39.500000	15.600000	190.000000	3550.000000
50%	44.500000	17.300000	197.000000	4050.000000
75%	48.600000	18.700000	213.000000	4775.000000
max	59.600000	21.500000	231.000000	6300.000000

Then we focused our target variable, ***species***, and observed that it has somehow a "balanced distribution", with 146 Adelie, 119 Gentoo, and 68 Chinstrap. Then, we see the pair-wise distribution (w.r.t. species) of our attributes (Figure 4).

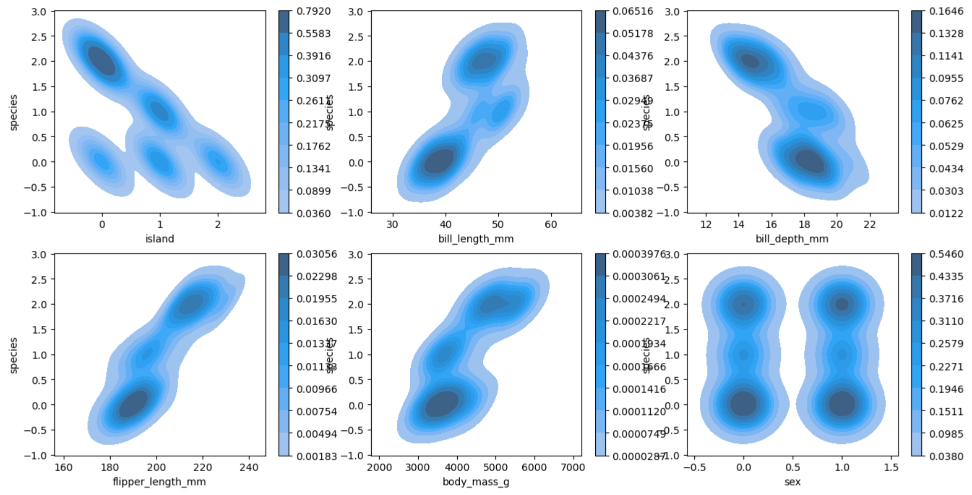

Figure 4: Attributes Distribution

2.1.2 Single Decision Tree

We first solved this problem using the single decision tree method. We change the **max_depth** and **min_samples_split**. And use other default settings, like Gini impurity criterion, none random state, etc.

Here we just show one example decision tree (Figure 5).

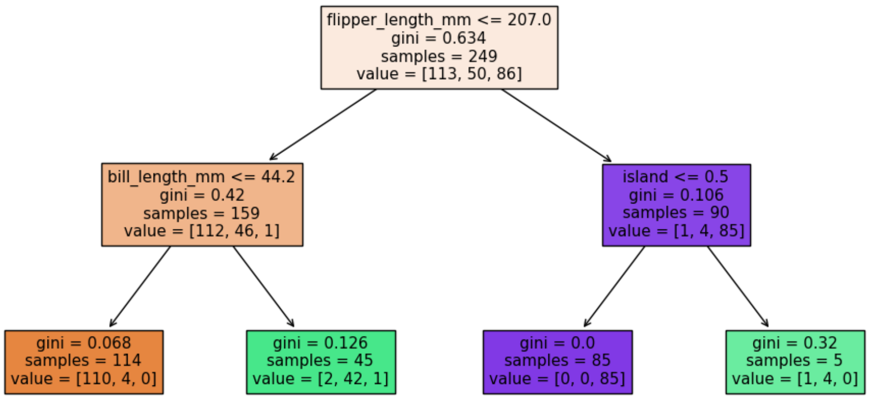

Figure 5: Decision Tree (max_depth=2, min_samples_split=3)

Our target is to make the impurity zero, i.e., there's only one class in one node. However, since we have set the max_depth in this decision tree, it cannot go further.

We changed hyperparameters, and run the decision tree model several times. We have the following results (Table 3).

Table 3: Decision Tree Accuracy Scores

max_depth	**min_size**	**train_score**	**test_score**
2	3	0.963855	0.952381
2	5	0.959839	0.940476
2	7	0.951807	0.904762
4	3	0.983936	0.964286
4	5	0.967871	0.940476
4	7	0.951807	0.904762
6	3	0.983936	0.976190
6	5	0.967871	0.940476
6	7	0.951807	0.904762

Here are some observations and explanations:

- When we increase **the max_depth**, the decision tree has more space to divide the samples into different classes. And hence the accuracy score increases.
- When increasing **the min_size**, it may cause impurity. Since there are more

samples in one node, it is more probable that there are different classes in this node.

2.1.3 Bagging of Trees

Now it comes to the bagging method. We created some bootstrap samples to obtain several diverse datasets. Then, we fit the decision tree for each resampled data. Finally, we calculate the mean derived from each tree to have the final results. Below are the results (Table 4).

Table 4: Bagging Trees Accuracy Score

max_depth	**num_tree**	**train_score**	**test_score**
2	3	0.963855	0.952381
2	5	0.959839	0.952381
2	10	0.963855	0.964286
4	3	0.995984	0.976190
4	5	0.995984	1.000000
4	10	0.995984	1.000000
6	3	0.995984	0.976190
6	5	0.995984	0.988095
6	10	0.995984	0.988095

Here are some observations:

- When increasing **the number of trees**, it's more likely that the training and testing performance increases, or for some cases, the performance does not change (like $(2, 3)$, and $(2, 5)$, where the testing score does not increase.)
- When increasing **the maximum depth**, as the reason discussed in Section 2.1.2. And in this scenario, the training accuracy does not improve when the depth is larger than 4. This might be because the tree has derived a good separation in the training process. There's no need to go deeper.

2.1.4 Random Forest

In the Random Forest model, we change the following hyperparameters in each trial. 1) the number of trees, 2) m values, i.e., the number of features to consider when looking for the best split. Bagging of trees is more like a data-level model, since it grasps subsamples and takes the majority of results. And Random Forest is more like a combination of data-level model and attribute-level model, since it introduces the randomness into data and attributes.

Table 5: Random Forest Accuracy Score

num_tree	**m_value**	**train_score**	**test_score**
1	2	0.987952	0.976190
1	4	0.983936	0.904762
1	6	0.983936	0.904762
3	2	0.987952	0.904762
3	4	1.000000	0.964286
3	6	0.991968	0.916667
5	2	1.000000	0.976190
5	4	0.991968	0.904762
5	6	1.000000	0.964286

Report Accuracy Scores Here's the table showing the results of the RF model (Table 5). Here are some observations regarding Table 5:

- When increasing **the number of trees**, the performance increases. See Section 2.1.3 for the reasons.
- When increasing **the value** m, i.e., the number of features considered in choosing the best split, the accuracy score decreases. This is because when including more features, the correlations between trees also increase, and this might affect the accuracy.

Report Bias and Variance We define $\hat{\theta}$ to be the point estimator, and the bias and variance can be calculated as follows:

$$\text{Bias} = E[\hat{\theta}] - \theta \qquad \text{Var}(\hat{\theta}) = E[(E[\hat{\theta}] - \hat{\theta})^2]$$

We change the number of trees in each trial, and want to find out the relationship between bias and variance with the number of trees. Below is the result (Figure).

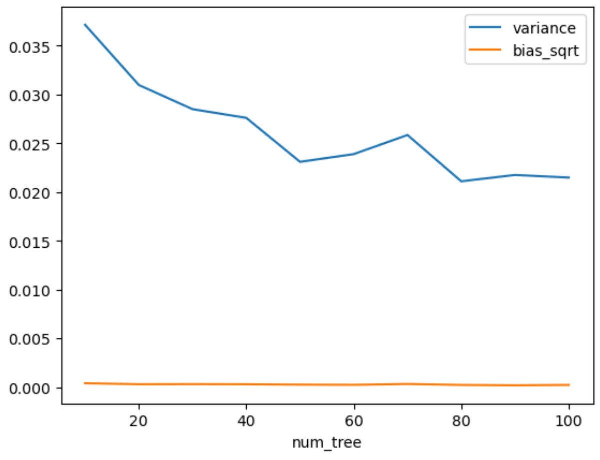

Figure 6: Relationship between Bias^2 and Variance with # trees

We have the following observations:

- **Variance**: When including more trees in the RF model, there is more randomness included in this model, at the data-level and attribute-level. And hence, taking the majority can bring us a more accurate result. Therefore, the variance decreases as the number of trees increases.
- **Bias**: Since bias estimates the distance between the estimated value and the true value. Here in these trials, we do not change our base learner, and hence the point estimator does not variate. Therefore, it seems that the bias term is irrelevant with # trees.

2.2 Multi-layer Perceptron Classifier

We have the Fashion-MNIST dataset, with 60,000 training samples and 10,000 testing samples. Each sample is a 28*28 image, with a label from 10 different classes. The image is flattened into a vector. Then, we use MLPClassifier to classify the data.

2.2.1 Hyperparameters Settings

hidden_layer_sizes This hyperparameter contains two pieces of information, the number of hidden layers, and the number of nodes in each hidden layer. # layers ranges from 1 to 3, and # of nodes is chosen from $\{50, 200, 784\}$

solver We choose Adam and SGD as our solvers here, where SGD denotes the stochastic gradient descent method. Adam is a stochastic gradient-based optimizer, which works pretty well on relatively large datasets (from the sklearn document).

learning_rate_init This hyperparameter controls the step size in updating the weights. The learning rate is chosen from $\{0.001, 0.01, 0.05\}$

2.2.2 Results and Analysis

I listed all the results in Appendix A. Here I just show some key observations and findings:

- **Solver**: From Figure 7-9, we can see that the accuracy score from the SGD solver is always less than that of the Adam solver. This might be because we choose different learning rates. When the initial learning rate is high, SGD will not converge sometimes, but Adam tends to converge faster. And when the learning rate is low enough, we can observe that, the accuracy score of SGD is approximately equal or even higher than Adam. This is because, SGD is more generalized, and hence can produce a more optimal solution.

- **Hidden size**
  - **# nodes** (Figure 8) Within the same hidden layer group, we observe that, increasing the number of hidden nodes will increase the testing performance. It's often the case that in training, we have a few more hidden nodes in each layer, to capture and store the additional weights. And it's also easy to prune them later on. And the rule of thumb is, we usually choose as many hidden nodes as dimensions needed to capture 70-90% of the variance of the input data set.
  - **# layers** (Figure 7) Since we only choose the layers from 1 to 3, it's unlikely to have the overfitting problem (but still we observe some). And increasing the number of layers seems to have no significant effect on Adam solver, but affects SGD a lot. For the training set, adding more layers decreases the training error. But in the testing set, adding more layers decreases the accuracy score, i.e., the generalization error increases (See Column 3 in Figure 7).
- **Learning rate**: (Figure 9) Learning rate denotes the step size in each iteration. We observe that, increasing the learning rate decreases the accuracy scores in the testing set. There's a significant decline in accuracy when the learning rate is larger than 0.01, which means the model can learn a little under these rates.

A Results in MLPClassifier

A.1 Accuracy Scores Table

hidden_size	optimizer	learning_rate	train_score	test_score
(50,)	adam	0.001	0.97355	0.878
(50,)	adam	0.01	0.91565	0.8677
(50,)	adam	0.05	0.8522333333333330	0.8421
(50,)	sgd	0.001	0.8944	0.8775
(50,)	sgd	0.01	0.9588833333333330	0.8815
(50,)	sgd	0.05	0.9751833333333330	0.8738
(200,)	adam	0.001	0.9980833333333330	0.897
(200,)	adam	0.01	0.9347666666666670	0.8741
(200,)	adam	0.05	0.8654666666666670	0.847
(200,)	sgd	0.001	0.9086333333333330	0.8872
(200,)	sgd	0.01	0.9907	0.8957
(200,)	sgd	0.05	0.9998166666666670	0.8961
(784,)	adam	0.001	0.9916333333333330	0.9036
(784,)	adam	0.01	0.9373833333333330	0.8735
(784,)	adam	0.05	0.84445	0.8264
(784,)	sgd	0.001	0.9158333333333330	0.8914
(784,)	sgd	0.01	0.9987833333333330	0.9031
(784,)	sgd	0.05	1.0	0.9083
(50, 50)	adam	0.001	0.9785666666666670	0.8787
(50, 50)	adam	0.01	0.9320833333333330	0.8797
(50, 50)	adam	0.05	0.7873333333333330	0.7808
(50, 50)	sgd	0.001	0.9151166666666670	0.885
(50, 50)	sgd	0.01	0.9757833333333330	0.8778
(50, 50)	sgd	0.05	0.9708166666666670	0.8766
(200, 200)	adam	0.001	0.9966666666666670	0.8995
(200, 200)	adam	0.01	0.9247	0.8772
(200, 200)	adam	0.05	0.73875	0.7285
(200, 200)	sgd	0.001	0.93285	0.8911
(200, 200)	sgd	0.01	1.0	0.8982
(200, 200)	sgd	0.05	0.9884833333333330	0.8944
(784, 784)	adam	0.001	0.9951166666666670	0.9071
(784, 784)	adam	0.01	0.9133833333333330	0.875
(784, 784)	adam	0.05	0.7760666666666670	0.7722
(784, 784)	sgd	0.001	0.95845	0.8991

(784, 784)	sgd	0.01	1.0	0.9055
(784, 784)	sgd	0.05	1.0	0.9129
(50, 50, 50)	adam	0.001	0.9817166666666670	0.8783
(50, 50, 50)	adam	0.01	0.9166	0.874
(50, 50, 50)	adam	0.05	0.8021	0.7971
(50, 50, 50)	sgd	0.001	0.9232	0.8888
(50, 50, 50)	sgd	0.01	0.9785833333333330	0.8796
(50, 50, 50)	sgd	0.05	0.9668833333333330	0.879
(200, 200, 200)	adam	0.001	0.9877	0.8976
(200, 200, 200)	adam	0.01	0.9086833333333330	0.8731
(200, 200, 200)	adam	0.05	0.5866166666666670	0.5784
(200, 200, 200)	sgd	0.001	0.9660666666666670	0.891
(200, 200, 200)	sgd	0.01	0.9905166666666670	0.8935
(200, 200, 200)	sgd	0.05	0.9903833333333330	0.8968
(784, 784, 784)	adam	0.001	0.99345	0.9063
(784, 784, 784)	adam	0.01	0.9051833333333330	0.879
(784, 784, 784)	adam	0.05	0.8412833333333330	0.832
(784, 784, 784)	sgd	0.001	0.9908833333333330	0.902
(784, 784, 784)	sgd	0.01	1.0	0.9087
(784, 784, 784)	sgd	0.05	1.0	0.9149

A.2 Visualizations

Notes: The yellow line represents the testing score from the Adam solver, while the blue one shows the score from the SGD solver.

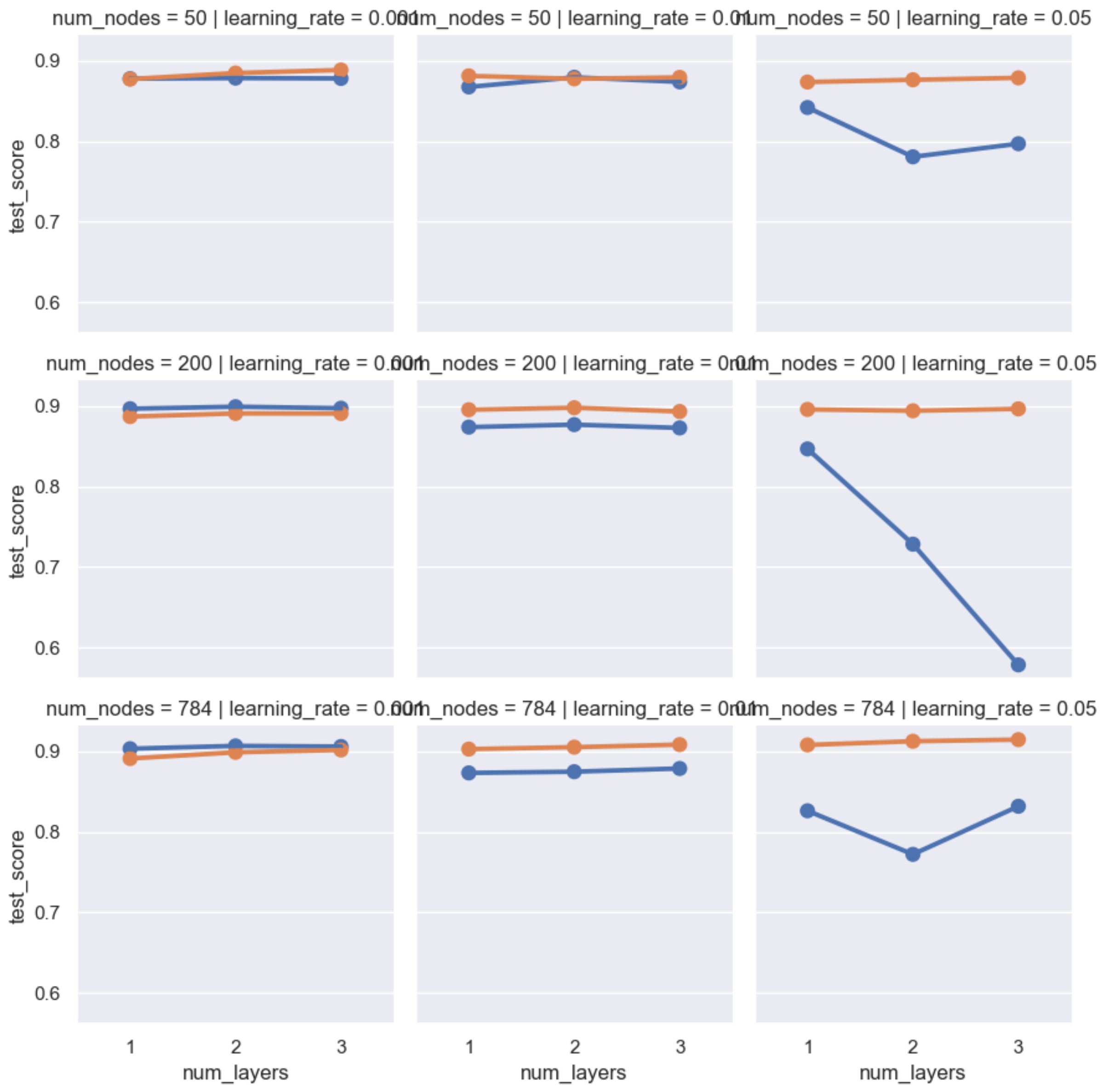

Figure 7: Relation between # layers and testing score, controlling # nodes and learning rate

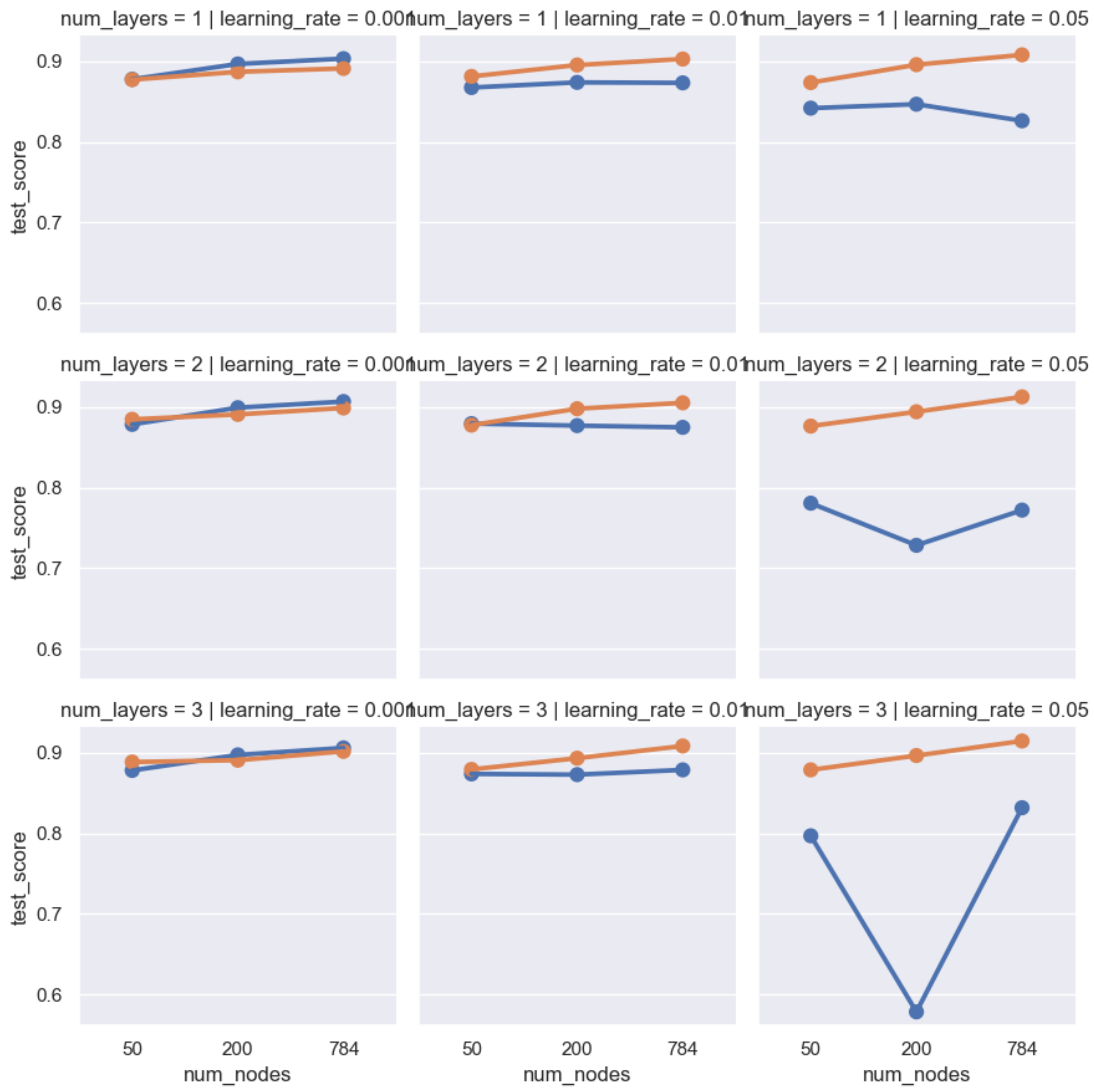

Figure 8: Relation between # nodes and testing score, controlling # layers and learning rate

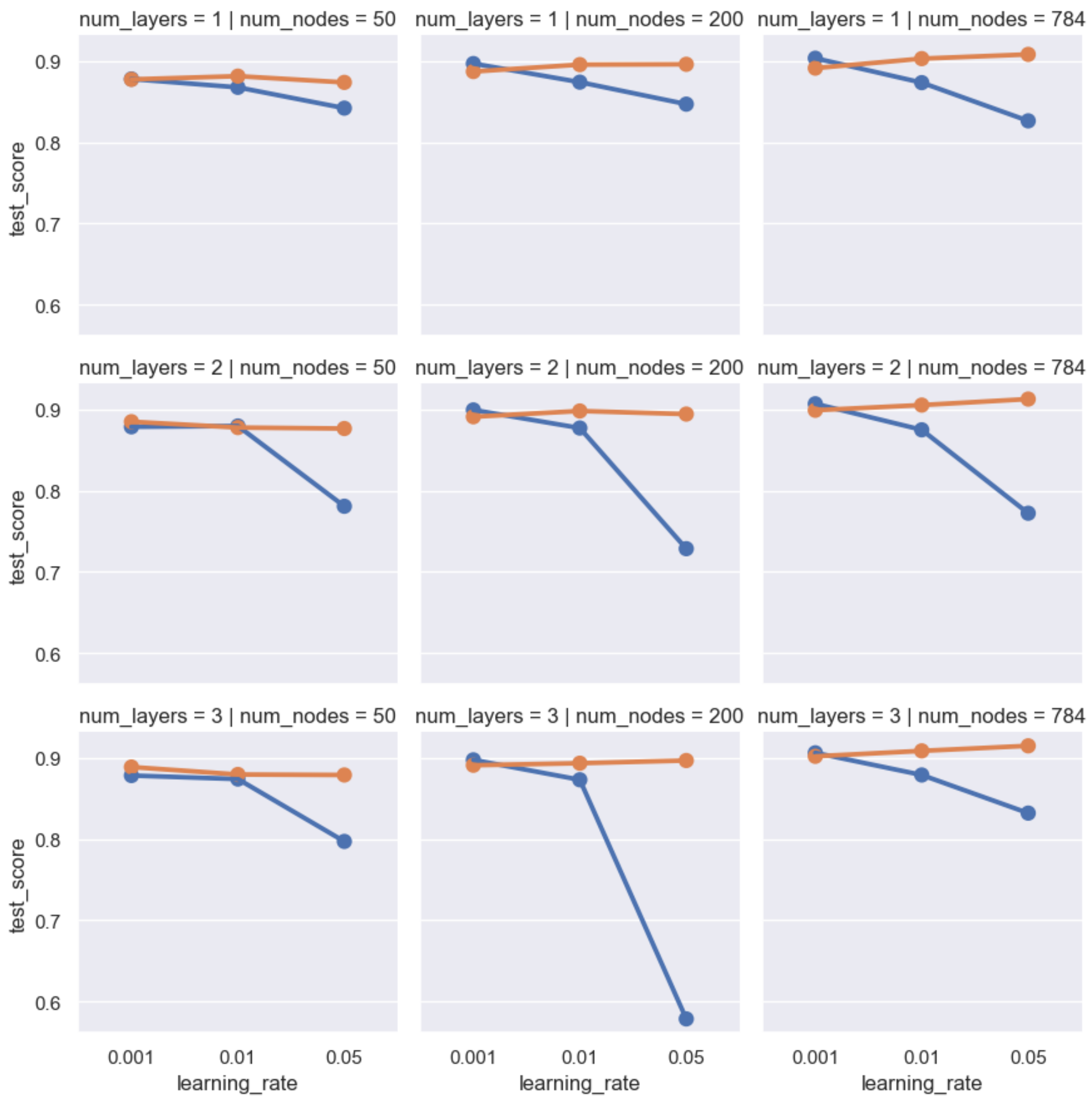

Figure 9: Relation between learning rate and testing score, controlling # nodes and # layers